DOCUMENT RESUME

ED 273 265 IR 012 271

AUTHOR Ward, Peggy M.
TITLE The Computer and Its Functions; How to Communicate with the Computer.
PUB DATE [85]
NOTE 20p.
PUB TYPE Guides - Classroom Use - Materials (For Learner) (051) -- Reports - Descriptive (141)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Computer Literacy; *Computers; Input Output Devices; *Programing; Programing Languages
IDENTIFIERS BASIC Programing Language; COBOL Programing Language; FORTRAN Programing Language

ABSTRACT

A brief discussion of why it is important for students to be familiar with computers and their functions and a list of some practical applications introduce this two-part paper. Focusing on how the computer works, the first part explains the various components of the computer, different kinds of memory storage devices, disk operating systems, and the peripheral devices used to communicate with other computers or the outside world. The second part provides a brief overview of programming languages and programming procedures, which includes a review of advantages and disadvantages of the BASIC language and general guidelines for writing computer programs. Examples are given in BASIC, including programs for performing a bubble sort and a binary tree sort. Flow charts are provided for the two types of sort programs as well as explanations of some of the BASIC codes. (DJR)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED273265

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
This document has been reproduced as received from the person or organization originating it.
Minor changes have been made to improve reproduction quality.
Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

PART I: THE COMPUTER AND ITS FUNCTIONS

by Peggy M. Ward

The computer has undeniably made an unprecented impact on the world. For the student, computers are becoming more and more a necessity for a well-rounded and complete education. In the future there will likely be few career fields that are not either directly or indirectly affected by the computer. It then becomes imperative for all students to at least become familiar with the "buzzwords" or jargon and the operation of the keyboard and its functions.

The range of possible uses for the student is numerous, but a few practical applications may be:

(1) Computer familiarity

(2) Word Processing

-term and research paper preparation

-chart and graphics preparation

-organization of notes

(3) Research

-analysis of experiments

-scheduling of projects

(4) Recreation

-games

-graphics design

-art work

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

Peggy M. Ward

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

There are infinte uses and applications for the computer, limited only by the student's imagination. Once the basic operation

IR012271

of the computer is learned, the student can work at his or her own pace. A whole new world of creativity may be opened to the student when they are allowed to develop to their fullest potential.

Some of the major definitions, functions, and components used in the computer world are listed below.

First of all, a computer is a device containing electronic circuitry which can perform computations, such as logical decisions and arithmetic operations without human intervention. It is not some super-human device that can outthink human intellect or perform in a judgmental way. If we fail to communicate correctly with it, the computer will not break down or become damaged, but we may fail in getting it to do what we want it to. There is a rule of thumb known in the computer world as GIGO or "Garbage In, Garbage Out".

Computations are performed by the execution of a sequence of directions in a given order. The directions or commands are really instructions that the computer has been preprogrammed to accept. They perform functions, such as arithmetic functions (add, subtract, divide, find the square root, etc.)

Computer processing normally consists of three parts:

(1) input data - information entered into the computer,

(2) process the input data, and

(3) output the data resulting from the processing of the input data.

THE MAIN COMPONENTS OF THE COMPUTER

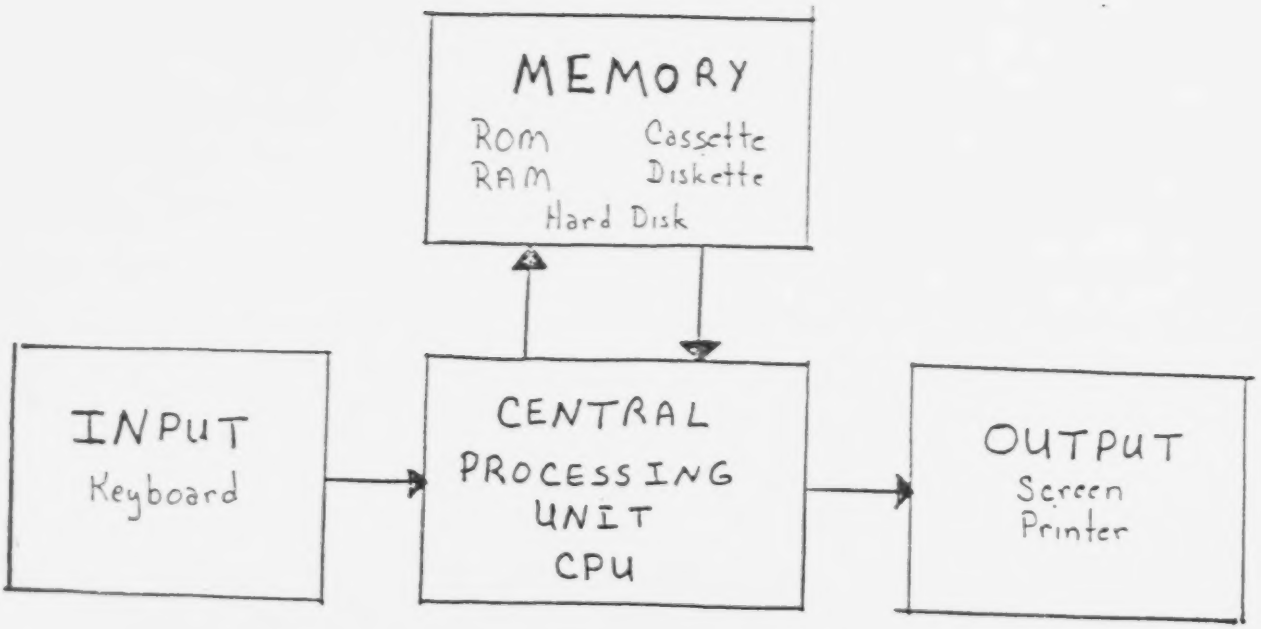

Fig. 1

Components of the Computer: The heart or center of all computer operations as pictured in Fig. 1 is the Central Processing Unit (CPU) which performs arithmetic calculations and makes logical decisions. The CPU must have access to storage units where data can be stored or retrieved from.

The most frequently used input unit is the keyboard, although other storage units, such as diskettes, may also be used. The input unit allows the user to enter information to the computer. The CPU processes the input data using memory for temporary storage and retrieval. And finally, the output unit allows the user to see the results of the program execution. The video monitor or screen gives a soft copy of the results that are lost when the power is turned off. The printer gives a more permanent type copy known as a hard copy.

Memory: Memory can be contained within the computer itself or externally in a separate unit connected to the computer by an interface. Five of the most frequently used types are listed below. The first two types are internal and the last three may be internal or external to the computer.

(1) ROM: The read-only memory or ROM is data stored into memory when it is manufactured and can only be read from and used. It remains permanently fixed in memory and cannot be altered. The initial computer start-up and other housekeeping functions are a part of the ROM's functions.

(2) RAM: The most commonly accessed memory storage is random access memory or RAM and can be read from, but unlike ROM, can also be written to. However, RAM is volatile and is erased when the power to the computer is off.

Microcomputers are commonly advertised as having so many "K" bytes in RAM; a "K" is 1,024 bytes. Today, a popular-sized small microcomputer may have a 64K memory; the computer actually has 64 time 1,024 bytes for a total of 65,536 bytes storage capacity.

(3) Cassette Recorder: The cassette recorder is the least expensive of the auxiliary or external memories discussed here, and is the same type used to record music and conversations. The recorder's main drawbacks are the length of time it takes to load and save programs, the limited capacity of the tape, and their somewhat limited reliability.

(4) Diskette Drives: A diskette drive, or "floppy" disk as it is better known, records data on a magnetically-coated, plastic disk much like a small flexible phonograph record, and may reside internally or externally to the computer. The floppy disk has the advantage of being many times faster with a greater storage capacity for data than does the cassette tape. For all of the advantages of the disk drive as compared to the cassette recorder, the disk drive is well worth the slightly higher cost. Because of the speed, capacity, reliability, and cost, the disk drive is the most preferred auxiliary memory unit in the small microcomputer world.

(5) Hard Disk Drive: The hard disk drive or Winchester drive stores data on a hard platter-type disk resembling a 33-1/3 rpm phonograph record in a sealed cartridge. The hard disk may have one or more layers of platters with a 1 to 50 megabyte capacity, and may also reside internally or externally to the computer. Although the costs of these drives has drastically decreased in recent months, they are still quite expensive when compared to the initial cost of buying a computer with other storage media.

Operating System: An operating system or supervisor is a computer program needed to coordinate all the functions and the flow of the activities between the CPU, the main and auxiliary memories, the video monitor, and all other peripheral or external devices connected to the computer.

Peripheral Devices: Devices connected to the computer are known as peripherals and help the computer communicate with other computers or the outside world. There are many devices available on the market today and more being introduced at a fast rate. Such devices include floppy and hard disk drives, daisy-wheel and dot matrix printers, modems, graphics screens, light pens, plotters, music and voice synthesizers, and temperature probes, just to mention some of the current available possibilities.

Most of the functions of these devices are self-explanatory, except perhaps the modem. With the modem you have the capability of connecting your computer to other computers via your telephone for interchange of programs and data with other users. It can be one of the most valuable peripherals to the computer user.

PART II: HOW TO COMMUNICATE WITH THE COMPUTER

Just as human beings communicate with one another so we must learn to "talk" to the computer in a language it understands. There are hundreds of computer languages on the market today and more on the way. Each language was developed for a specific purpose; for instance, FORTRAN, derived from FORmula TRANslation, is a powerful language especially useful in the scientific and engineering field where there are very small or large numbers and lots of number crunching. COBOL or COmmon Business Oriented Language is the most commonly used language for business, and was designed to deal with financial and management applications. The two languages just mentioned are what is known as higher level languages, but the language that the computer better understands is machine language or low level language. Machine language is both hard for the computer programmer to understand and extremely difficult and time consuming to write programs for.

The language most familiar to small microcomputer users is BASIC (Beginner's All-purpose Symbolic Instruction Code) developed by John Kemeny and Thomas Kurtz of Dartmouth College in the early 1960s as an easy-to-use language for those with little or no computer programming experience. BASIC was originated for its simplicity but has evolved through the years as a very powerful and highly flexible language.

BASIC programs are a sequence of ordered instructions to perform tasks or solve a problem like adding many numbers together, drawing

pictures, or playing games. The following procedures may make it easier to write a program with fewer errors in it:

(1) Think the problem or task through in logically ordered steps. Be sure of what your input will be and what you expect for output.

(2) Write the steps down in English-like sentences; these written steps are known as an algorithm. By following the logic of the algorithm, the sequence of steps to solve the problem should become apparent.

(3) Write a corresponding set of instructions in BASIC being generous with remarks. It may be hard to remember several days or weeks later what your code is doing without reminders. Number the BASIC statements in increments of 10 to allow for statements to be inserted in the program at a later time without having to renumber all the statements. Organize the set of instructions into small modules whenever possible; these may be tested individually before they are incorporated into the whole program.

(4) After the coding is finished, don't be afraid to play computer with your program. Go through the program line by line, processing the data just as the computer would. This may seem like a lot of unnecessary work, but it will save considerable time, especially if your computer access time is limited.

(5) Enter the coded program, run it, and correct your mistakes, if any. Don't be discouraged if there are mistakes or bugs in your program; all programs written of any substantial size will have bugs, and lots of them.

(6) Time permitting, test all probable combinations of input data on your program to make sure all situations are covered. Of course, this may only be feasible on smaller programs.

Disadvantages of BASIC: Although the BASIC language is one of the easiest higher level languages to understand, there are two disadvantages that limit its use in highly technical and time-limited applications; these are speed and, to a lesser degree, structure. When attempts are made to correct these shortcomings, one nearly always is a trade-off for the other, as shown below.

Structure: Structure is the disadvantage that can be more easily corrected. With very short programs, structure is not much of a problem, but when the programs become very complex, they tend to become more difficult to: (1) plan and write an algorithm for; (2) correct errors; and (3) read and understand what the program is really doing. With structuring, some of the pitfalls like unreadable, unmaintainable, skip-all-over-the-place programming may be avoided. Here's some steps that may help:

(1) Write your program in a "top-to-bottom" fashion, organizing it to always flow downward. Avoid jumping all over your program; it is not only hard to follow now, but will also be hard to modify later.

(2) Begin with a "shell program" that depicts the main decision-making parts of your program and fill in the functional parts one at a time after they are tested.

(3) Program in modules, if possible. Write each module, test it throughly to see if the desired results are obtained, and insert it in the program.

(4) Use subroutines where and when the same series of code will be used more than once at different points in your program. These help make your program more readable, save memory and programming time, make the program more flexible, and may greatly simplify debugging and testing.

After writing many and more complex programs, each individual programmer tends to develop their own programming style and logic. Some of these help the programmer identify functions within the program because they are always located at certain places or in certain ways in the program. For example, line numbers may signify a specific area of function, shown as follows:

Line No.	Used For
1-99	program header, name, date created, modifications performed and dates, variable identifications, any comments about the uniqueness of the program
100-199	initialization of dimension statements and variables, start-up routines,
200-999	main body or shell of program
1000-99999	subroutines, and you can even break the type of subroutine into divisions here.

Another example is to use the same variable names in all your programs having the same functions, like I, J, and K for the index of FOR loops.

(5) Be sure there are ample remarks. More memory will be used and it might take longer to type the program in, but you will have a program that is instantly understandable by you, as well as anyone else that reads your program. You will also have a program that is of greater ease to change in the future, as well as being attractive as a printer listing.

Speed of Execution: The second aspect is by far the more complex to understand. First of all, we have to talk about the way a program is executed after you type in "RUN". Most microcomputers come initially equipped with a higher level language usually known as "interpreted" BASIC.

Higher level computer languages are either interpreted or compiled. Interpreted programs are executed a line of code at a time, breaking down each line every time it is executed during the program to the machine level that the computer best understands. If the program uses the same code fifty times, then it must be interpreted fifty times. It is no wonder interpreted BASIC is slow.

Whereas a compiled program or source code using a higher level language is entirely transformed into a complete machine level language program known as object code or machine code before the program is run never having to be interpreted again unless changes are made to the source code. The computer no longer has to interpret each line every time it is executed whether it is used one time or a hundred times.

As almost always happens in the computer world, however, when speed is gained, memory may be sacrificed and vice versa. A compiled

program uses many times the amount of memory that an interpreted program does; it must store the expanded and linked object code and other housekeeping functions. Also, the compiler applications program itself it much more complex and memory using. Just a note here, compiled BASIC is available, but the program is costly.

There are techniques to enhance the speed of interpreted BASIC, but some of these techniques are considered to be bad programming practices. As stated before, readability, documentation, and attractiveness may be compromised if you want your BASIC programs to run faster.

Here are some techniques to use in your program, if speed of execution is your goal:

(1) Use multiple statements per line. Actually you can use whole lines as mini-modules. In particular, a multi-statemented line beginning with an IF statement would be executed only if the first command in the line were TRUE.

(2) Delete spaces and remarks from your program, except where absolutely necessary for clarification.

(3) Use one letter variables, such as A, B, C, etc. Comparisons for long variable names are time consuming.

(4) Always initialize the most frequently used variables first in your program. When the computer searches the variable table where the variables are stored, it will not have to look very far.

(5) Delete unnecessary parentheses around arithmetic calculations. All separately enclosed numbers must be dealt with individually.

(6) Reuse variable and dimension statements in your program, if possible.

(7) Use integer variables (these hold values less than 65,536), where possible. It takes more than two bytes to store a larger number and much longer to locate the other bytes.

(8) Use dimensioned arrays sparingly. They take more time to access than other variables.

(9) Limit the number of subroutines used. The computer must search through your program for them once it encounters a GOSUB.

(10) If you do use subroutines, place them at the beginning of your program for the reason stated in the previous item.

(11) Use FOR. . . .NEXT loops wherever possible. They are faster than decision making IF statements.

(12) If you are an advanced programmer, you may want to experiment with some machine language routines inserted into your BASIC program.

(13) Data structures used in your program play an important part in how fast your program will run.

For example, sorting is the process of arranging a set of data items in either an ascending or descending order, but the techniques of sorting and the data structure used to sort the data can make a great difference in the amount of time it takes to complete the job.

A comparison between a Bubble Sort and a Binary Tree Sort shows this to be true. The ascending bubble sort procedure allows you to move the smallest number to the top through a series of comparisons and swaps of consecutive data items. It may take up to the number of data items minus one passes to complete the process. On each pass the next smallest number will always be passed or "bubbled" up to its final position in

the order list. Below is a program written to perform a Bubble Sort:

```
100  DIM C(100)
110  FOR I = 1 TO 100
120  INPUT "NUMBER PLEASE ⇒ ";C(I)
130  NEXT I
140'
200  DONEFLAG = 0
210  WHILE DONEFLAG = 0
220  DONEFLAG = 1
230  FOR I = 100 TO 1, STEP -1
240  IF C(I) < C(I-1), THEN SWAP C(I), C(I-1):DONEFLAG=0
250  NEXT I
260  WEND
270'
300  FOR I = 1 TO 100
310  LPRINT C(I)
320  NEXT I
330  END
```

Note the variable DONEFLAG. Initially DONEFLAG is set to 0 before the WHILE loop is entered. The DONEFLAG is then set to 1 within the loop and, if there is no swaps performed during that pass, all the data is ordered. Otherwise, the DONEFLAG is set back to 0 and the process continues.

By contract, the Binary Tree Sort divides the items being searched by approximately one-half on each pass; so even on the initial pass it

only looks at one-half, and then on the next pass, one-half of that portion. Both sorts will sort duplicate entries. Here is a program that demonstrates this procedure:

```
100  DIM N(100),C(100,2)S(20)
110  L=1:R=2
120  FOR I = 1 TO 100
130  INPUT "NUMBER PLEASE => ";N(I)
140  NEXT I
150'
200  FOR I = 2 TO 100
210  B = 1
220  IF N(I) > N(B) THEN A=B: B=C(A,R): IF B <> 0 THEN
                GOTO 220 ELSE C(A,R)=I: GOTO 240
230  A=B:B=C(A,L):IF B<>0 THEN GOTO 220 ELSE C(A,L)=I
240  NEXT I
250'
300  D=-1:I=1
310  D=D+1:S(D)=I:I=C(I,L):IF I <> 0 THEN GOTO 310
320  I=S(D):LPRINT N(I):D=D-1:IF D < 0 THEN GOTO 340
330  I=C(I,R):IF I <> 0 THEN GOTO 310 ELSE GOTO 320
340  I=C(I,R):IF I <> 0 THEN GOTO 310
350  END
```

You may compare these two techniques for length of execution time and it might surprise you to find that the Binary Tree Sort will run

approximately 10 times faster than the Bubble Sort. On my computer it took 70 seconds to sort 100 numbers using the Bubble Sort and only 7 seconds using the Binary Tree Sort.

Also notice the difference between the simplicity of the Bubble Sort and the longer, more complex Binary Tree Sort. To get a cleaner picture of the difference between the complexity of the Bubble Sort and the Binary Sort, compare the algorithms in Fig. 2 and Fig. 3. As stated earlier, there is almost always a trade-off between the amount of memory and programming time used, and execution time.

ALGORITHM FOR BUBBLE SORT PROGRAM

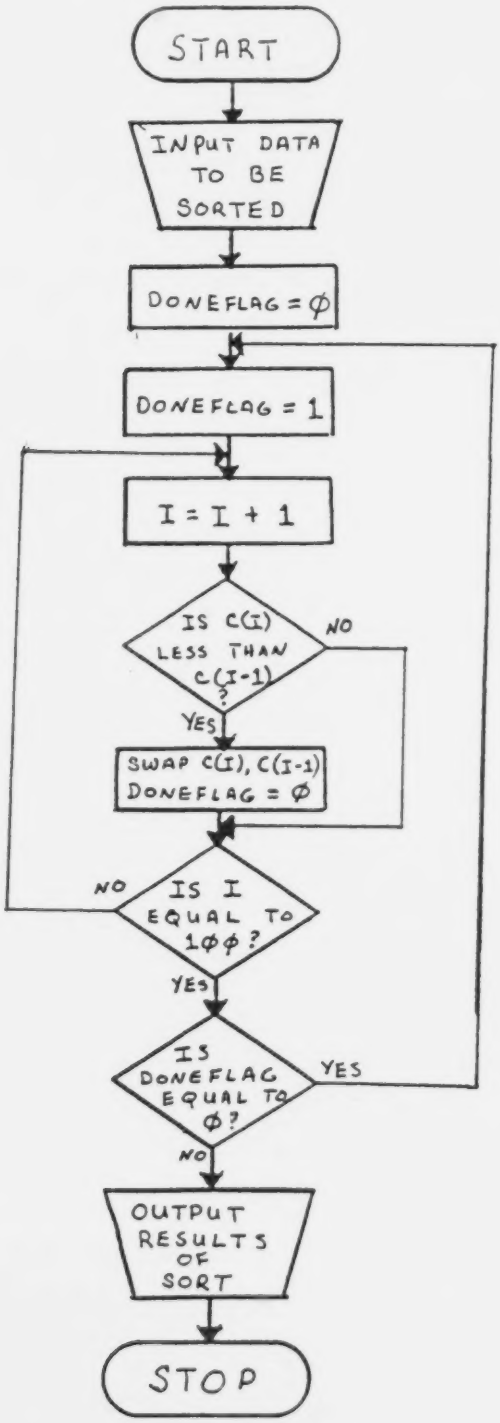

FIGURE 2

ALGORITHM FOR BINARY TREE SORT

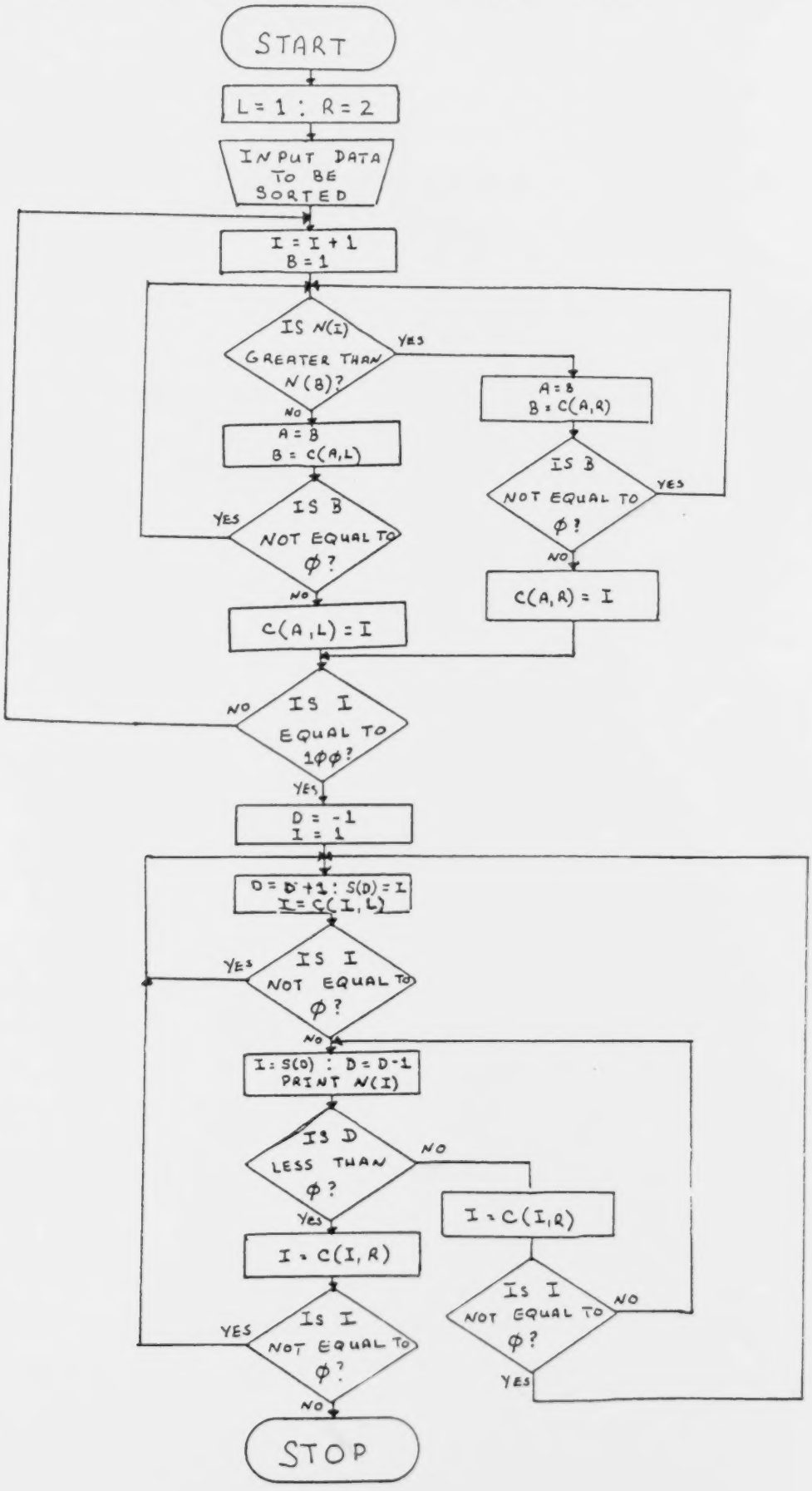

FIGURE 3

Although the above suggestions make program execution faster, they may produce poor programming habits and many should be avoided in most cases; good programming practices should be the first and foremost goal. Computer programming is a very exacting skill and must be implemented to attain the maximum in well-written, correctly executed programs.

Happy programming.